DOCUMENT RESUME

ED 276 418 IR 012 415

AUTHOR Weitzman, Louis
TITLE Designer: A Knowledge-Based Graphic Design Assistant.
INSTITUTION California Univ., San Diego, La Jolla. Inst. for Cognitive Science.
SPONS AGENCY Office of Naval Research, Arlington, Va. Personnel and Training Research Programs Office.
REPORT NO ICS-R-8609
PUB DATE Jul 86
CONTRACT ONR-N00014-85-C-0133
NOTE 29p.
PUB TYPE Reports - Evaluative/Feasibility (142)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Computer Graphics; *Computer Simulation; Decision Making; Design; *Expert Systems; *Man Machine Systems
IDENTIFIERS *Computer Assisted Design; Image Analysis

ABSTRACT

This report describes Designer, an interactive tool for assisting with the design of two-dimensional graphic interfaces for instructional systems. The system, which consists of a color graphics interface to a mathematical simulation, provides enhancements to the Graphics Editor component of Steamer (a computer-based training system designed to aid instruction in the domain of propulsion engineering) by allowing nonprogrammer users of the Graphics Editor to create interactive, dynamic views of the simulation. Designer consists of three interrelated processes, an Analyzer, a Critiquer, and a Synthesizer, which are coupled to a domain-dependent knowledge base consisting of design elements; design relationships; techniques for their identification; sets of constraints for establishing a context or style and for critiquing a design; and generative techniques for creating design alternatives. The Analyzer parses a design based on the elements and relationships of the domain and records this information in the knowledge base; the Critiquer uses this information along with domain-based design constraints to indicate where the current design succeeds or fails; and the Synthesizer uses knowledge representing the current state of the design to generate design alternatives within a design style. Subsystems incorporated into the system architecture include Steamer's frame-based knowledge representation facility and an assumption-based truth maintenance system. The text is supplemented with several graphic illustrations. A bibliography, an ICS Technical Report List, and a list of earlier reports by people in the Cognitive Science Lab are also provided. (Author/KM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☒ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED275418

DESIGNER: A KNOWLEDGE-BASED GRAPHIC DESIGN ASSISTANT

Louis Weitzman

July 1986

ICS Report 8609

DESIGNER: A KNOWLEDGE-BASED GRAPHIC DESIGN ASSISTANT

Louis Weitzman

July 1986

ICS Report 8609

NPRDC-UCSD Intelligent Systems Group
Institute for Cognitive Science
University of California, San Diego
La Jolla, California 92093

This research was sponsored by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-85-C-0133, Contract Authority Identification Number, NR 667-541. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government. Requests for reprints should be sent to Louis Weitzman, Institute for Cognitive Science, C-015; University of California, San Diego; La Jolla, CA 92093.
Copyright © 1986 by Louis Weitzman.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ICS 8609	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Cognitive Science University of California, San Diego 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Personnel & Training Research Programs, Office of Naval Research (Code 1142 PT)
6c. ADDRESS (City, State, and ZIP Code) C-015 La Jolla, CA 92093	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-C-0133
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61153N	RR04206	RR04206-0A	NR 667-541

11. TITLE (Include Security Classification)
Designer: A Knowledge-Based Graphic Design Assistant

12. PERSONAL AUTHOR(S)
Louis Weitzman

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO June 86	1986 July	19

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Graphic design, knowledge-based system, assumption-based truth maintenance systems.
05	08		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Designer is an interactive tool for assisting with the design of two-dimensional graphic interfaces for instructional systems. Graphic domain knowledge, stored in a frame-based representational facility, is coupled to a domain independent mechanism which analyzes and critiques the user's original design, and then synthesizes design alternatives. These alternative solutions are generated within a design context, or style, and are based upon graphic constraints. The underlying motivation is to improve the quality of the interfaces by making them more consistent and visually more effective.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Michael G. Shafto	22b. TELEPHONE (Include Area Code) 202-696-4596
	22c. OFFICE SYMBOL ONR 1142 PT

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Contents

OVERVIEW 2
DOMAIN-DEPENDENT KNOWLEDGE REPRESENTATION 4
- Elements 5
- Relationships 10
- Constraints 10
- Design Context 12

DESIGN PROCESSES 13
- Analysis 13
- Critique 13
- Synthesis 14

CONCLUSION 15
ACKNOWLEDGMENTS 17
BIBLIOGRAPHY 17

Designer: A Knowledge-Based Graphic Design Assistant

LOUIS WEITZMAN

Applying technologies from artificial intelligence and cognitive science to the development of computer-based training and computer aided design systems can provide support in areas where developers and users lack expertise. In addition, intelligent tools can substantially enhance the process of design. *Designer* is a tool to aid users of *Steamer's Graphics Editor*. Steamer is a computer-based training system to aid instruction in the domain of propulsion engineering (Hollan, Hutchins, & Weitzman, 1984). It is used to help students develop an understanding of the complex domain of steam propulsion. The system consists of a color graphics interface to a mathematical simulation. One can view and manipulate this simulation at a number of different hierarchical levels through the color interface. The current system contains over one hundred color views which range from abstract, high-level representations of the plant (Figure 1) to detailed views of gauge panels quite like the actual gauge panels in a ship (Figure 2). It was apparent that an editor for creating and maintaining this set of views was essential. The Graphics Editor allows nonprogrammers to graphically create interactive, dynamic views of the simulation. Figure 3 depicts the black-and-white interface of the Editor. This facility has allowed propulsion engineering instructors to create substantial portions of the student interface to this advanced training system. Even though the Editor was built for the construction of Steamer views, the tool is domain independent and has been used to build interfaces monitoring the real-time performance of a computer operating system.

Views are constructed out of graphic components called *icons* which represent elements in the steam domain. Icons perform two tasks. First, they graphically depict the state of the simulation. For example, pumps are red when stopped and green when operating, dials display their value by positioning an indicator, and pipes show their value by animating their fluid. Second, the user can affect the simulation via the icons. When the user positions a cursor over the icon and clicks the mouse, the state of the icon and its associated value in the simulation are modified. For example, a pump's state toggles from off to on and a dial's value is set by positioning the indicator. Figure 4 shows a sampling of the types of icons available to users of this Editor. In creating a view, the user selects the icons to be added to the view from a menu on the black-and-white screen. The user then positions and sizes the new icon on the color display. This icon has its parameters defaulted according to the type of icon chosen. Then, through a process of incremental refinement, the user modifies only those attributes that differ in this particular application.

It is unrealistic to assume that instructional designers are facile with graphic design. Facilities were built into the Graphics Editor to support the construction of *good* views. These facilities include various types of grid latching and the maintenance of aspect ratio for specific icons. However, these constraints were often overridden by the designer. Even working within these constraints, users of the Graphics Editor often violate important graphic design principles and have difficulty maintaining stylistic conventions across sets of views. Designer is a tool to enhance the Graphics Editor by supplementing the designer's domain knowledge with the necessary graphic expertise.

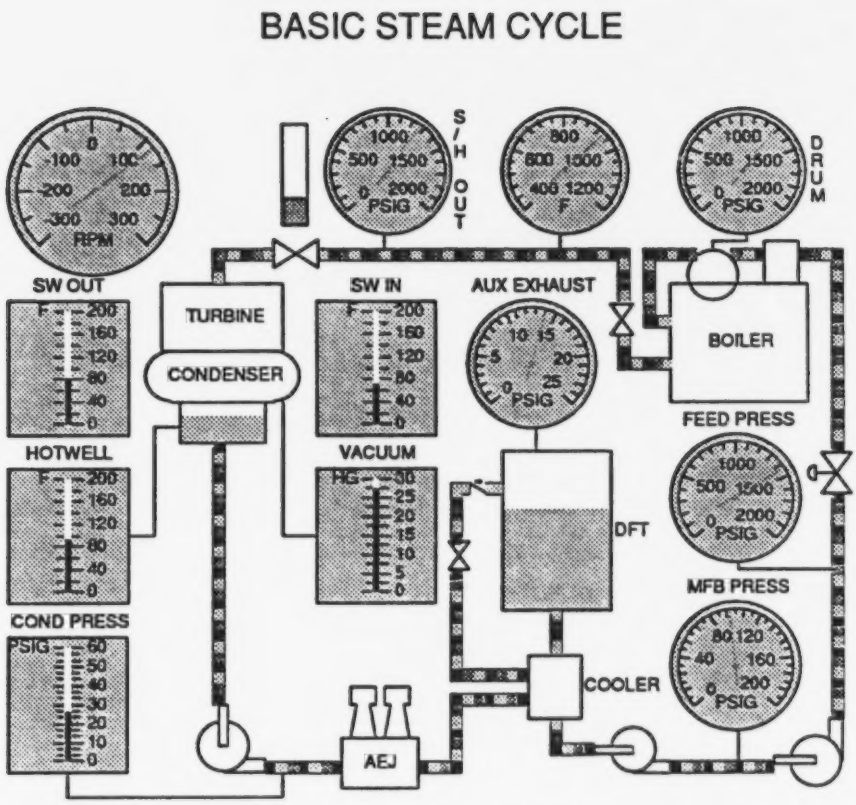

FIGURE 1. *Basic Steam Cycle.* The color interface can depict the simulation at many different levels of abstraction. This high-level, conceptual view illustrates the complete steam cycle.

OVERVIEW

Designer provides visual expertise to users of the Graphics Editor interactively constructing new Steamer views or modifying existing ones. It consists of three interrelated processes, an *Analyzer*, a *Critiquer*, and a *Synthesizer*, coupled to a domain-dependent knowledge base. This knowledge base consists of design elements, design relationships, techniques for their identification, sets of constraints for establishing a context, or style, for critiquing a design, and generative techniques for creating design alternatives. The Analyzer first parses a design based on the elements and relationships of the domain, and records this information in the knowledge base. The Critiquer uses this information along with domain-based design constraints to indicate where the current design succeeds or fails. Using knowledge representing the current state of the design, the Synthesizer then generates design alternatives within a design style. The separation of these three processes from the knowledge base provides independence and modularity to the system. This flexibility creates a technology that will be extensible to other design domains.

In order to support the internal mechanisms of Designer, a series of generic subsystems have been incorporated into the system architecture. These tools include Steamer's frame-based knowledge representation facility (MSG) for storing all domain knowledge, and an Assumption-Based Truth Maintenance System (ATMS) for maintaining alternative design decisions which define the design space.

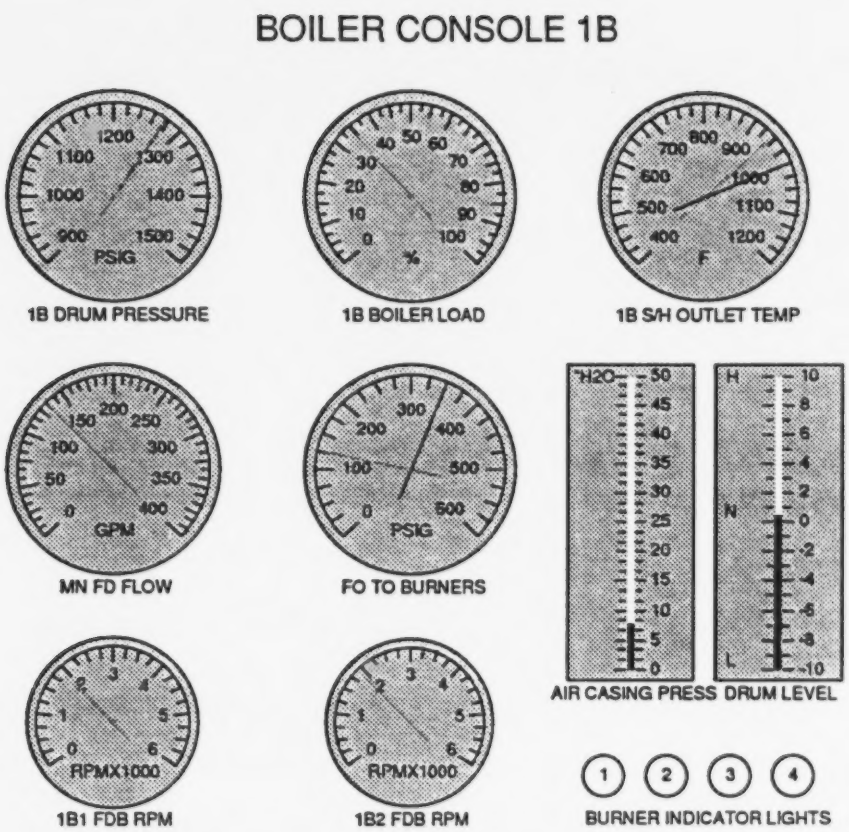

FIGURE 2. *Boiler Console 1B.* This view of the boiler console panel illustrates the color interface representing detailed views of actual engineering stations.

The system is being developed in the object-oriented programming environment of Flavors on a Symbolics 3600 family processor. The use of object-oriented programming techniques of Flavors has greatly facilitated the implementation and is used throughout the system. A preliminary interface used in the development of the system is shown in Figure 5. The multipaned interface provides access to existing Graphics Editor functions and new Designer functions through scrolling command panes (upper right collection of panes) while access to the domain knowledge is provided in a mouse sensitive graphing pane (upper left pane). A lisp interaction pane is provided (lower left pane) along with a scrolling pane for Designer information (e.g., constraint violations; lower right pane). Designer is just one application, or activity, in a larger instructional simulation environment (Hollan, Hutchins, McCandless, Rosenstein, & Weitzman, in press). Other activities in this environment include a model control facility, Steamer; a view construction facility, the Graphics Editor; a facility to create new icons with new behaviors, the Icon Editor; and a facility to create lessons for students based on particular views and simulation models, the Lesson Editor. A status line, which is consistent throughout all simulation activities, displays information relevant to the current activity. In Designer, the status line (near the bottom of the screen) displays the current values for the system, subsystem, view, and design style. The labels and their values are all mouse sensitive, providing access to functions on the class of item (clicking on the label) or operations on the item itself (clicking on the specific value).

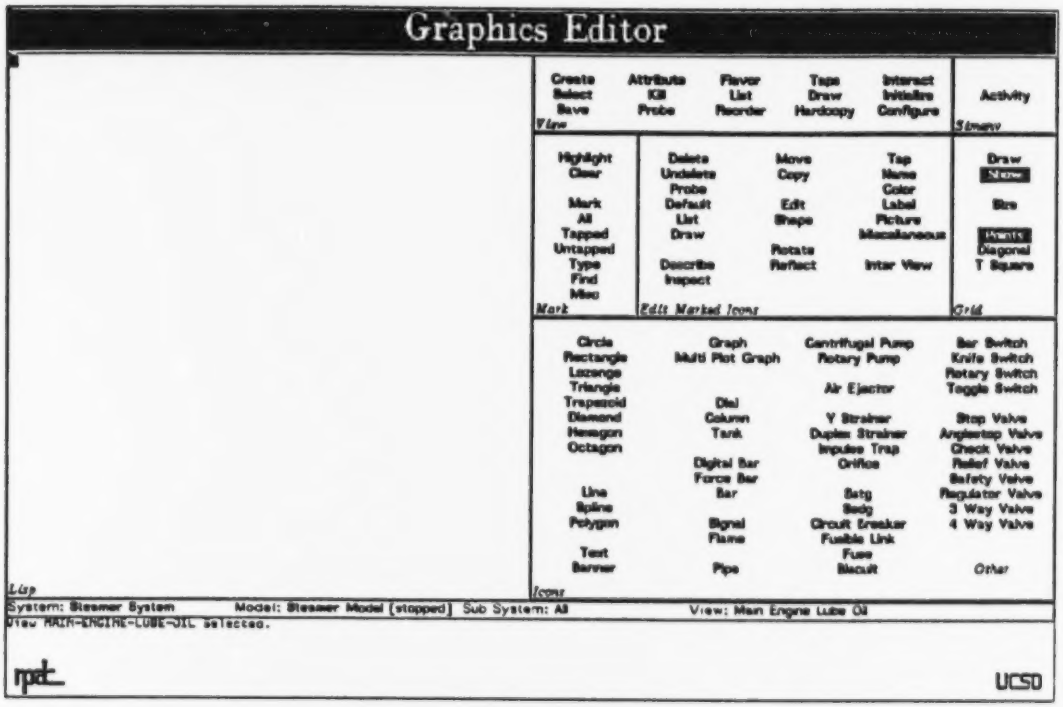

FIGURE 3. *Graphics Editor Interface.* The Editor is a domain-independent tool allowing nonprogrammers to create graphic interfaces for monitoring and controlling underlying simulations or real-time processes.

DOMAIN-DEPENDENT KNOWLEDGE REPRESENTATION

Much has been written about the knowledge required for graphic design (Bertin, 1983; Cheatham, Cheatham, & Haler, 1983; Ching, 1979; Dondis, 1973; Hurlburt, 1977; Marcus, 1986; Reilly & Roach, 1984; Sherwood, 1981; Taylor, 1960; Wong, 1972). Unfortunately, the literature does not suggest a consistent representation for this knowledge. Designer incorporates much of this knowledge and maintains it in the frame-based representational system, MSG. Designer concentrates on the knowledge describing the domain elements, their relationships, graphic constraints imposed on both the elements and their relationships, and graphic techniques for their modification. For general graphic design the elements refer to points, lines, planes, etc. (Wong, 1972). In Designer, however, these domain elements represent the icons contained within a Steamer view which are characterized by their graphic properties.

MSG, a *flavor enhancer*, provides a class structure on top of the Flavors object-oriented programming facility. It provides the ability to define classes of objects and create instances of those classes. Each class provides a set of attributes, or slots, that define the characteristics of the class. Slots are grouped together in roles. Slots inherited from the class' abstractions, or parent classes, are included with the locally defined slots to completely describe the class. When a new class is defined, an instance of a meta-class is created that will maintain all pertinent information about the new class. This includes how to create new class instances, where to store the new instances, how to manipulate them, etc. In addition, when a new class is defined, a new flavor of the same name is also defined. Instances of the class are actually instances of this new flavor with the instance variables corresponding to all slot

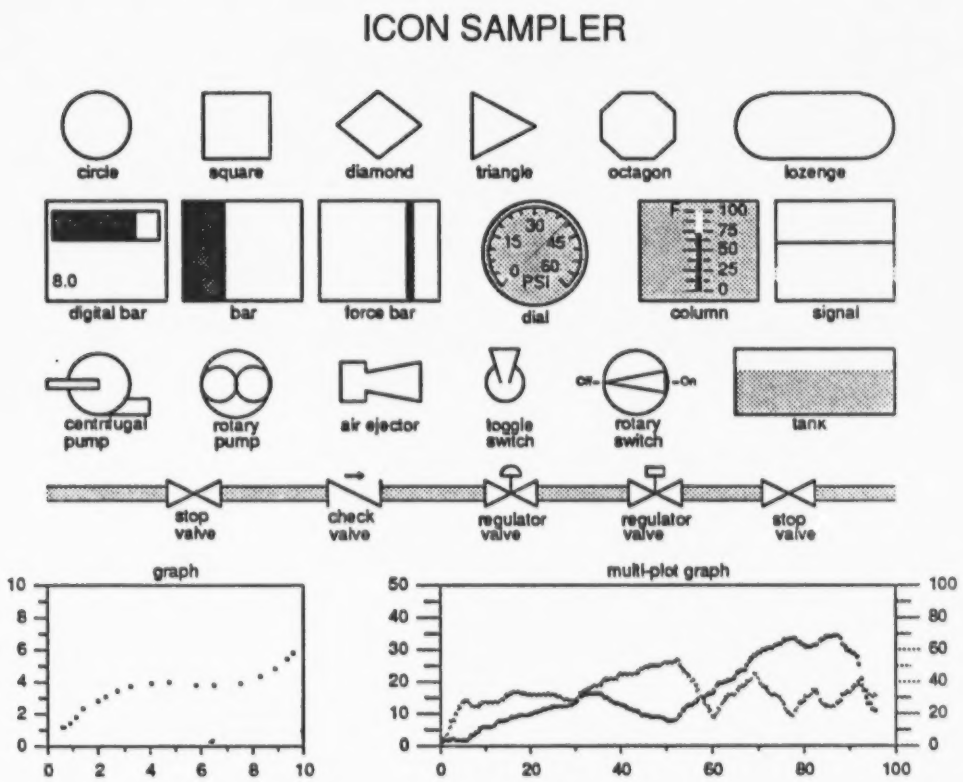

FIGURE 4. *Icon Sampler.* This view illustrates a sample of the graphic icons available to designers creating interactive interfaces.

attributes of the class. The new instances are stored on the class object. MSG can be used incrementally, so as new domain knowledge is defined and recorded in the knowledge base, this information will automatically be included in the analyses. Thus, as the system expands, the domain knowledge can easily grow. This ability to incrementally build the domain knowledge is important and increases the system's flexibility.

A tool to create, maintain, and inspect the knowledge base was incorporated into Designer. It provides a flexible graphing facility of the domain-dependent knowledge base. The structure of the graphic class hierarchy is clearly visible in the graphing window of Figure 6. The graph ranges from more abstract classes on the left to more specific classes on the right. The ability to edit and inspect classes and their instances can be accessed through mouse clicks and menu selections. The menu of commands to operate on the class, its instances, or its flavor is shown in Figure 7 for the class *elements*.

Elements

The MSG class of ***elements*** records all domain elements that will be used in subsequent design analyses. The following is the definition of this class which includes instance variables for a name,

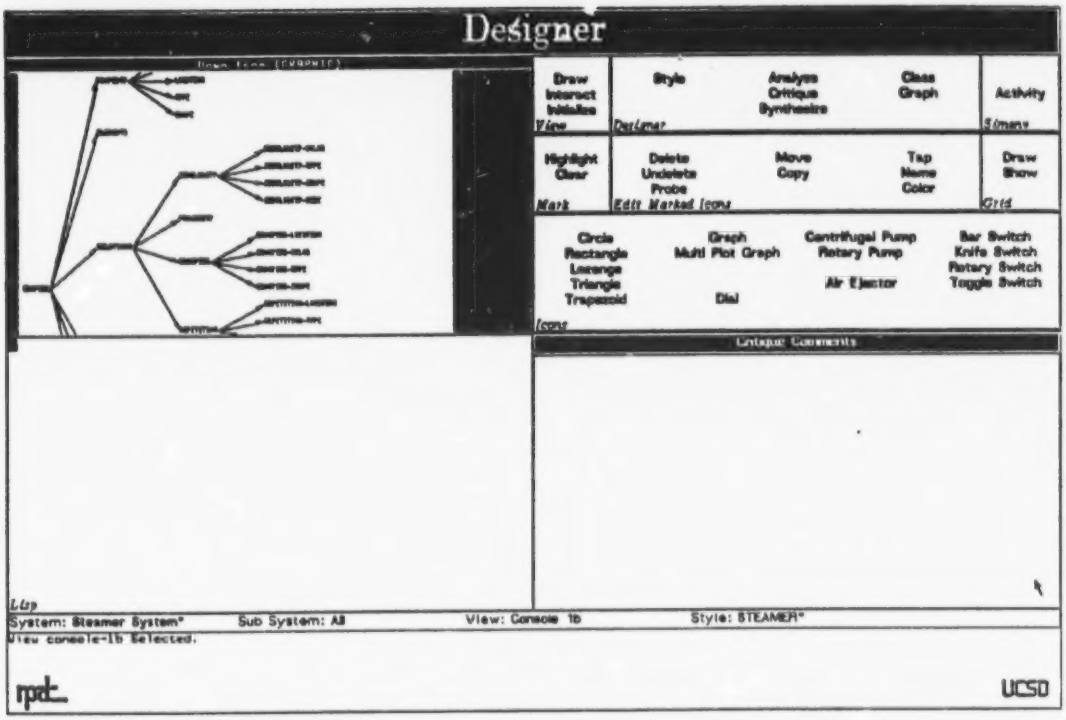

FIGURE 5. *Designer Interface.* The Designer interface provides access to all Graphics Editor commands while providing additional commands to control the design processes and related functions. Domain-dependent knowledge is displayed in a graphing window pane. State information (e.g., current values for system, subsystem, view, and style) is provided in the status line near the bottom of the screen.

a description, all roles (subdivided into slots), all abstractions (parent classes), all used-as-abstractions (classes that use this class as an abstraction), and all of the instances of the class in the current design.

#<CLASS ELEMENTS 46622062>, an object of flavor CLASS, has instance variable values:

NAME:	ELEMENTS
DESCRIPTION:	"a graphic element"
ROLES:	((PROPERTIES ((COLOR (A COLOR) NIL NIL) (SIZE NIL NIL NIL) (LOCATION NIL NIL NIL) (TYPE (A TYPE) NIL NIL) (SHAPE (A SHAPE) NIL NIL) (ELEMENT NIL NIL NIL))))
CONNECTIONS:	NIL
SYNONYMS:	NIL
ABSTRACTIONS:	(GRAPHIC)
USED-AS-ABSTRACTION:	NIL
INSTANCES:	(#<ELEMENTS DIAL-1 44645216> #<ELEMENTS DIAL-2 44644710> #<ELEMENTS DIAL-3 44644670>)

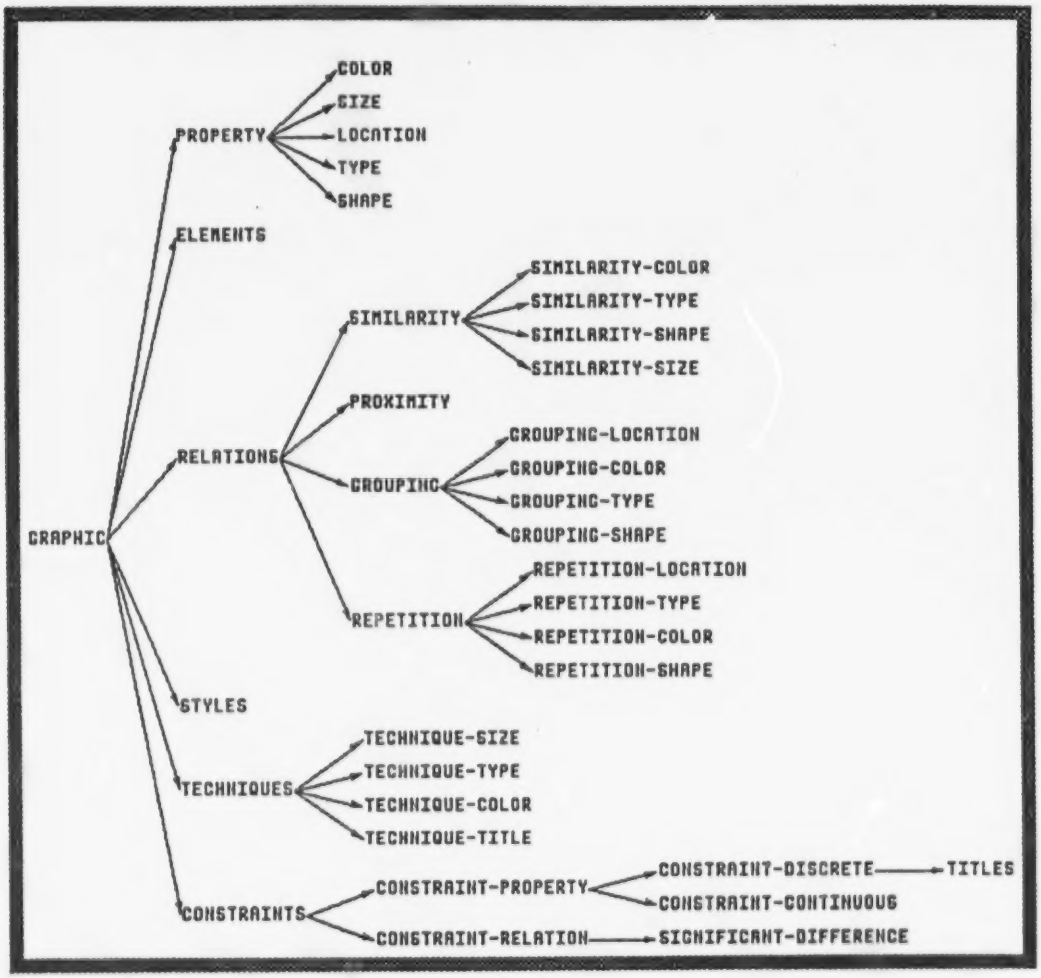

FIGURE 6. *Domain Knowledge Base.* A graphing tool aids the creation and maintenance of the domain knowledge represented in a frame-based system. Each node in this graph represents a class in the domain of graphic design. Class inheritance is immediately apparent with classes changing from abstract to more specific as one moves through the graph from left to right.

The slots of this class include graphic properties used to distinguish the elements. These are the graphic properties of color, size, location, type, and shape. The values of these properties on an instance are in fact instances of other MSG classes (see Figure 6) that represent valid values for the class. For example, the class of *color* includes instances for Steamer's basic colors. The class *size* includes instances describing a range of sizes from very-small to very-large, while the class *shape* includes instances of basic geometric shapes like linear, circular, rectangular, etc. Figure 8 illustrates the current set of instances for the classes *color*, *size*, *type*, and *shape*. In addition, there is a class slot to store the domain element an instance of this class will represent.

In the above example, three instances of the class *elements* are stored on the instance variable *instances*. All three of these objects represent dial icons in the current view. One of these three objects representing a small, blue dial is shown below.

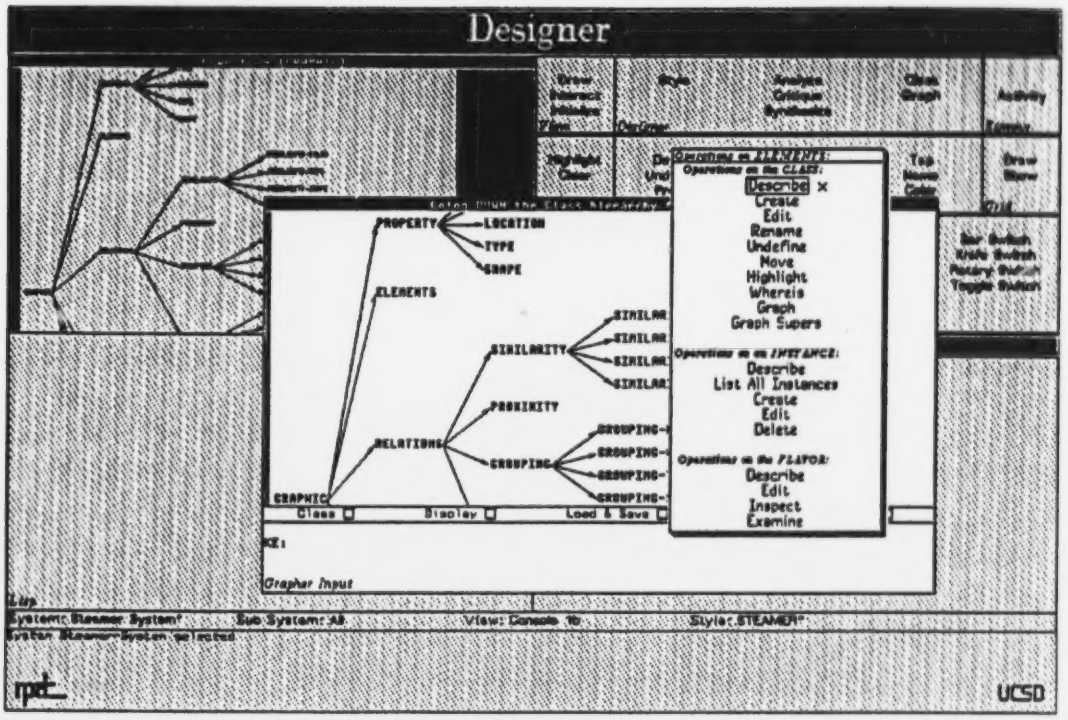

FIGURE 7. *Operations on Domain Knowledge.* The mouse sensitive graph nodes provide access to operations on the class, its instances, and its flavor definition through mouse clicks and menu selections. The menu is presenting operations for the selected class *elements*.

#<ELEMENTS DIAL-1 44645216>, an object of flavor ELEMENTS, has instance variable values:

IDENTIFICATION:	DIAL-1
STRING-FOR-PRINTING:	NIL
COLOR:	#<COLOR BLUE 44644212>
SIZE:	((:X #<SIZE SMALL 44644633>) (:Y #<SIZE SMALL 44644633>))
LOCATION:	((:X 0.846) (:Y 0.521))
TYPE:	#<TYPE DIAL 44644216>
SHAPE:	#<SHAPE CIRCULAR 44644224>
ELEMENT:	#<DIAL 44644230>

This example (of a class definition and description of one of its instances) illustrates that all class slots (i.e., color, size, location, type, shape, and element) become instance variables on the flavor representing the class. These variables have been initialized on the actual instances to the appropriate class values (e.g., the *blue* instance of class *color* is stored on the *color* instance variable, the actual domain element the instance represents is stored on the *element* instance variable).

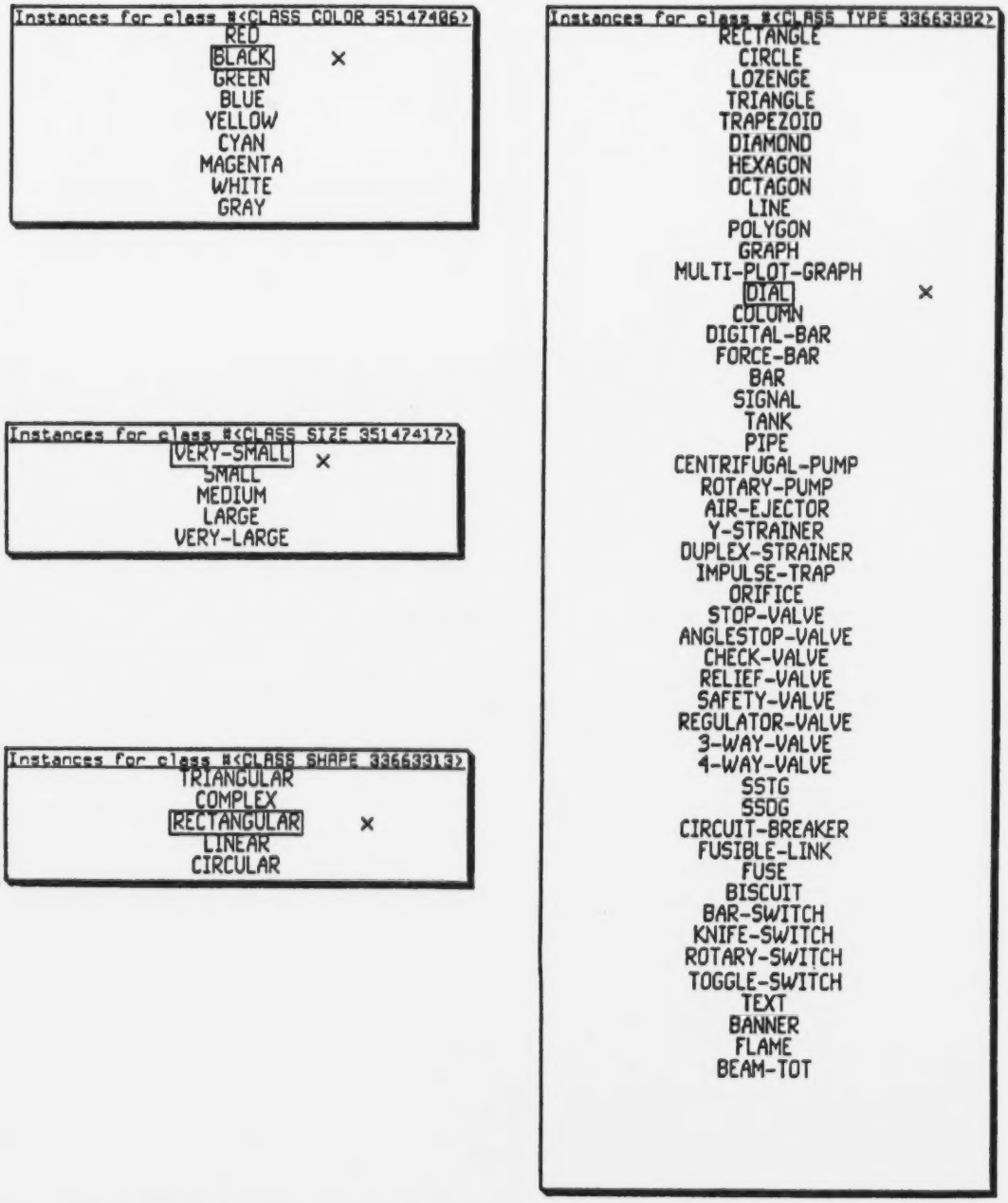

FIGURE 8. *Instances of Graphic Property Classes.* Menus list the instances of classes representing the four graphic properties *color*, *size*, *shape*, and *type*.

Relationships

Currently the graphic relationships in the knowledge base are *similarity, proximity, grouping,* and *repetition.* As can be seen in Figure 6, the relationships of similarity, grouping, and repetition are further classified by the graphic properties of the elements (e.g., grouping by color, repetition by type, etc.). The following is the description of the MSG class of similarity of color:

#<CLASS SIMILARITY-COLOR 46622203>, an object of flavor CLASS, has instance variable values:

NAME:	SIMILARITY-COLOR
DESCRIPTION:	"the graphic relationship for the similarity of color"
CLASS-VALUE:	COLOR
NUMBER-FOR-NAME:	0
ROLES:	((PROPERTIES ((ELEMENTS NIL NIL NIL) (STATE NIL NIL NIL) (CERTAINTY NIL NIL NIL))))
CONNECTIONS:	NIL
SYNONYMS:	NIL
ABSTRACTIONS:	(SIMILARITY)
USED-AS-ABSTRACTION:	NIL
INSTANCES:	(#<SIMILARITY-COLOR SIMILARITY-COLOR-BLACK 44645357> #<SIMILARITY-COLOR SIMILARITY-COLOR-BLUE 44645350>)

In the above example, there are two instances of the relation class *similarity-color*, one for black elements and one for blue elements. The instance representing the relation of similarity of color blue is illustrated below. Here, the previously described dials appear since they all have a blue face color. These elements are stored on the instance variable *elements.*

#<SIMILARITY-COLOR SIMILARITY-COLOR-BLUE 44645350>, an object of flavor SIMILARITY-COLOR, has instance variable values:

IDENTIFICATION:	SIMILARITY-COLOR-BLUE
STRING-FOR-PRINTING:	NIL
ELEMENTS:	(#<ELEMENTS DIAL-1 44645216> #<ELEMENTS DIAL-2 44644710> #<ELEMENTS DIAL-3 44644670>)
STATE:	NIL
CERTAINTY:	:HIGH

All relations know how to handle a generic message to identify occurrences in the design of the relation that they represent. When an occurrence is identified, a new instance of the class is created, stored on the class object, and initialized with all the elements participating in the relation. Relations can also build on one another. For example, elements in proximity to one another may form grouping relations, and groupings may form repetition relations (depending on the elements properties and their layout).

Constraints

Domain constraints consist of both basic graphic design *principles* important in the construction of two-dimensional views and view *standards* that are adopted for the current application. Principles are those constraints that transcend view sets and are generally accepted methods of making images consistent, unambiguous, and visually effective. The principle of Significant Difference of Size, for instance, states that when elements are a different size, they should be significantly different so as not

to create a sense of ambiguity (Sherwood, 1981) (Figure 9). Elements that are larger represent objects that are more important or physically larger in the real world. If they are not, they should be the same size as other, similar elements in the view. In Figure 1 of the Basic Steam Cycle, the dial indicating RPMs is significantly larger than the others, denoting the fact that it is the most important dial of the set. This principle manifests itself not only in terms of the size of elements but also in terms of the other properties of the elements. The principle in terms of location, for instance, tends to align elements unless there is a reason (of importance or physical fidelity) to accentuate the differences in location. Therefore, in the knowledge base there exist multiple instances of the MSG class of significant difference. Graphic design standards differ from principles because they are special constraints that tend to exist for a given set of designs for a given application. In Steamer, the use of a title is a standard. The restriction of pipes to be within an acceptable range of sizes is another standard.

Constraints can also be categorized as restrictions on properties of elements or restrictions on their relationships. Constraints on properties take the form of discrete constraints, restricting a property to be a specific value, or continuous constraints where the value can range between a minimum and a maximum value. An example of multiple discrete constraints is the standard of Title. Three constraints on the elements properties of *type*, *size*, and *color* restrict the values to be *text*, *large*, and *yellow*, respectively. The standard of Pipe Size is a continuous constraint where the value is restricted between a minimum and maximum size.

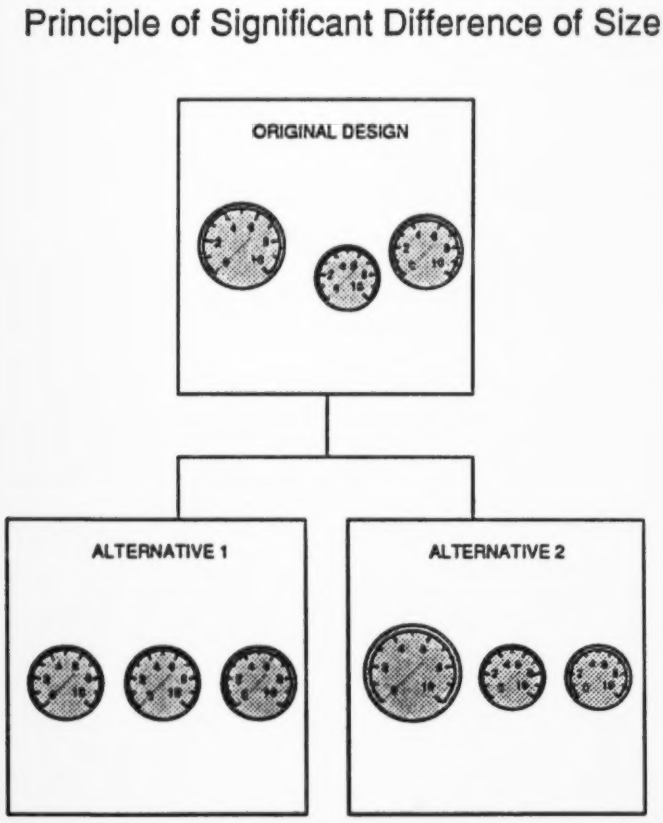

FIGURE 9. *The Principle of Significant Difference of Size.* This design principle states that when elements are a different size they should be significantly different so as not to create a sense of ambiguity. Given an original design consisting of three dials, two alternatives are presented from a larger solution space. The first alternative suggests no difference in importance and therefore no difference in size. Alternative 2 takes into account the fact that the right two dials are grouped together. The other dial, being physically separate and larger, may be perceived as more important and therefore should be *significantly* larger.

Design Context

A design should be sensitive to the context in which it is created. It is this context that defines the external constraints which shape and guide the final solution. In Designer, this context is referred to as a *style* and is constructed by selecting those constraints (principles and standards) that are to be enforced within this context. Good design in one style may not necessarily be good design in another. Modifying the style within which a critique is made ultimately affects the final form of the design.

A graphic style is also defined by the visual techniques employed in the communication of information (Dondis, 1973). These visual techniques represent a vocabulary in which to describe the design. These techniques in conjunction with the constraints may suggest a variety of graphic procedures to modify an alternative. These procedures are similar to Mittal, Dym, and Morjaria's (1986) *design methods*. For example, the visual technique of *Regularity* may take on a value of regular, neutral, or irregular, each suggesting alternatives consistent with its definition. Highly regular designs will accentuate similiarities of elements and relationships, while irregular designs accentuate the differences. It is the constraints that indicate a discrepency in the design, while the interaction of the techniques suggest the graphic procedures (maybe more than one) that will modify the design. Figure 10 illustrates a style editor which allows the user to name a style, select graphic constraints to be active within it, and choose values for the visual techniques by selecting the appropriate value.

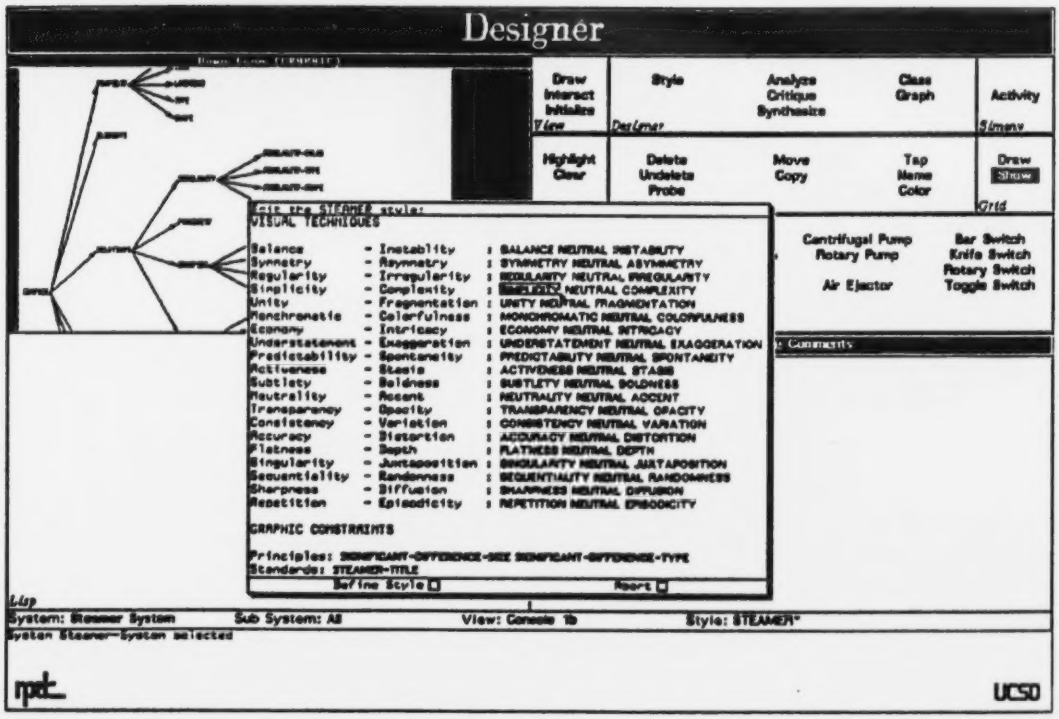

FIGURE 10. *Style Editor.* This menu edits the graphic style, or context, in which a design critique occurs. A style is defined by the graphic constraints (i.e., principles and standards) that are active and the values chosen for the visual techniques. These visual techniques in combination with the constraints generate the graphic procedures for modifying the design.

BEST COPY AVAILABLE

DESIGN PROCESSES

Design involves a cycle of gathering information, making decisions based on that information, and reviewing the consequences of those decisions. New information gleened from this process is incorporated back into the cycle for subsequent refinement of the design. This analysis-synthesis-review cycle is a general process used in all design whether it be for computer interfaces, industrial applications, or architecture. The process is domain independent.

In order for Designer to carry out the analysis phase of the process, two steps must take place. First, the system must parse the design into domain elements and relationships. This is Designer's *Analysis* process. Second, the system must locate areas that need to be improved. This is the *Critique* process. Together, these processes represent the analysis phase of the generic design process. Once the first two steps have occurred, the system is ready to suggest alternative procedures for modifying the design. This is Designer's *Synthesis* process.

Since the overall goal is for the system to be an online assistant and not assume control, review occurs interactively with the user selecting and confirming decisions presented by the system. Information is incorporated back through the process as output from one cycle becomes the input for the next cycle of critique. An alternative approach (Brown & Chandrasekaran, 1986), which may be incorporated in future versions of the system, is to provide a *plan* which specifies the order in which design steps are invoked by the motivating techniques. Each of Designer's processes is described in more detail below.

Analysis

The analysis process parses the design and locates existing domain elements and relationships. Identifying the elements is straightforward because of the use of icons in the Steamer interface and the object-oriented nature of their implementation. An instance of the MSG class *elements* is created for each icon. The elements instance variables are appropriately initialized with values for each property being an instance of the corresponding MSG class.

Once the domain elements have been created, the system locates instances of domain relationships. This task is easy for people but very difficult for computers. Much work has been done in the area of image analysis, but seldom with the goal of *beautifying* drawings (Pavlidis & Van Wyk, 1985). To maintain the independent nature of the analysis, generic messages are sent to each relation class to identify instances of the class within the design. When an occurrence is found, an instance of the MSG relation class is created and initialized. This includes the recording of the elements that participate in the relation on the appropriate MSG slot.

Critique

As Christopher Alexander suggests, the notion of a *misfit* is more compelling than a *fit* and is a driving force behind the ultimate shape of a design (Alexander, 1974). In Designer, the misfits are identified as violations of the domain constraints. The Critiquer creates a *comment* for each unsatisfied design constraint within the current style. These critique comments are Flavor objects that store their underlying constraint and the elements involved in the violation. Descriptions and justifications of the comments that are based on this constraint can then be presented. These comments, displayed in the scrolling pane of the black-and-white interface, are mousable and can be highlighted (graphically highlighting those elements involved) and described in terms of their underlying principle or standard. Critiques themselves are implemented as flavor objects that store the object being critiqued (the Steamer view), the style in which the critique takes place, and a list of all the relevant comments for this object and style. Figure 11 illustrates a critique based on the principle of the Significant Difference of Size of three dials shown in the original design of Figure 9. Two violations of this principle, one for

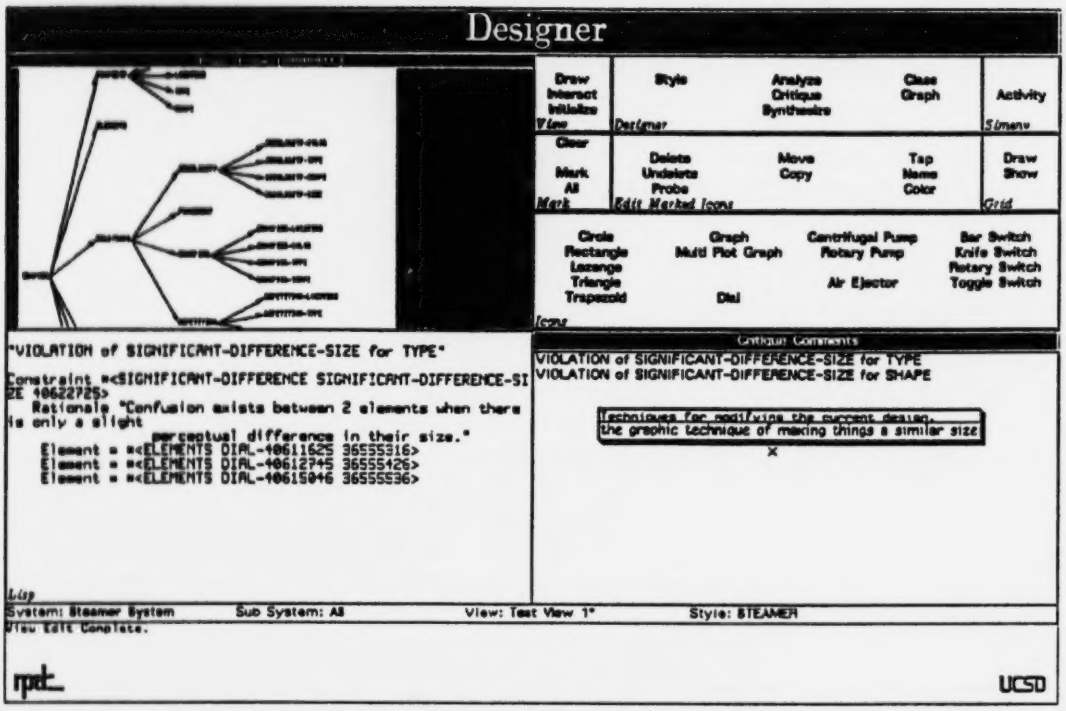

FIGURE 11. *Critique of Three Dials.* The Designer interface illustrates a critique in progress of the original design from Figure 9. This critique, being executed in the *Steamer* style, has generated two critique comments based on the principle of Significant Difference of Size. One comment refers to elements of a similar type while the other refers to elements similar in shape. A description of one violation is presented in the lisp pane and a menu of techniques for modifying the design based on this violation is presented in a pop-up menu.

similar typed elements and one for similar shaped elements are displayed in the scrolling pane. A description of the first violation is presented in the lisp pane.

It thus becomes possible under this paradigm to request multiple critiques, each based on a different style. This is an especially powerful paradigm for views that may need to be presented in different media, each with different constraints. For example, a style appropriate for a high resolution color display may not be appropriate for a black-and-white hardcopy presentation where features are not as clearly distinguishable.

Synthesis

Design decisions are made in the synthesis phase in order to incrementally refine the elements and their relationships. Knowledge of the elements and their relationships along with the comments from the critiquing phase forms the basis for these design modifications. Each comment communicates to the constraint on which it is based via generic messages in order to determine the graphic procedures for satisfying the existing violation. More than one procedure may be available to satisfy the constraint and all possiblities are presented to the user. These procedures are a result of the interaction of the various visual techniques and the design constraints which describe the style.

Once the user decides to remedy a given comment by clicking on it with the mouse, various graphic procedures are presented in a pop-up menu (Figure 11). These procedures will all modify the design in

order to satisfy the constraint, but will do so differently. Since there is no *correct* solution these procedures represent alternatives and there is no attempt to suggest that one alternative may be better. The variation of alternatives are based on the definition of the style's visual techniques. For example, if a style is defined to be *simple* and *regular*, the constraint of Significant Difference of Size will generate a very different solution than if the style is defined as *complex* and *irregular*. Figure 12 illustrates alternative solutions in three different styles all defined with the constraints of Significant Difference of Size and of Location. The only difference between the styles is the articulation of the visual techniques ranging from simple and regular (Style 1, Figure 12A) to complex and irregular (Style 3, Figure 12B). With the same initial design, each style creates different solutions. These solutions satisfy the constraints but are based on varying procedures of generation from the defined visual techniques. In Style 1, the system looks for the simplest most regular solution possible. This results with all dials in each solution being the same size and aligned on an axis (similar location). Style 3, on the other hand, has chosen the opposite approach where no two dials in the final solution are the same size and no alignment occurs. Style 2 (Figure 12C) takes a more moderate approach with two distinct (and significantly different) dial sizes and some alignment. This simple example illustrates how the interaction of the graphic constraints with the visual techniques will generate alternative solutions.

These alternatives are maintained by a new form of truth maintenance system, an ATMS (De Kleer, 1986a, 1986b, 1986c). With the ATMS multiple alternatives are maintained and can be explored simultaneously. Unlike previous truth maintenance systems which just manipulated justifications, this system additionally manipulates assumption sets. As a result, inconsistent information can exist and it is possible to work effectively and efficiently in the problem space. Context switching is free, and most backtracking and all retraction is avoided. In Designer, the assumptions that are manipulated are the alternatives created by incremental design decisions. Solutions at any stage in the design process are the consistent, noncontradictory environments maintained by the ATMS. Any contradictions that arise are handled by the ATMS and will not appear in the same environment.

This new form of truth maintenance system is well suited for tracking multiple alternatives in the design space where a reasonable number of the potential solutions must be examined. Designer interacts with this system by creating an ATMS class for each domain element. Whenever an element is modified, a new ATMS node is added to this class. These classes represent the different alternatives of the original domain element. Multiple nodes coexist in the solution space but only one will be present in any ATMS solution. The justifications of what style is current and what constraint generates the modified element can be added to these nodes to further restrict the space of valid solutions.

Because context switching is free, the user can explore the design space by interactively inspecting the individual ATMS environments. Each solution can be displayed on the color screen and explain itself in terms of the underlying assumptions and justifications. Based on these assumptions and justifications, an alternative can describe its derivation and individual decisions can be described in terms of their potential contribution to a final solution. The system thus conveys design precepts while the user is viewing a specific instantiation of a design alternative. Thus, the user's knowledge of constructing visual presentations is enhanced for future design.

CONCLUSION

There exists a number of interesting research projects similar in nature to Designer. They are all knowledge-based systems providing an environment to aid the creation and verification of design alternatives. Some provide exploratory environments in well-defined domains (e.g., Palladio, for circuit design [Brown, Tong, & Foyster, 1983]) while others, like Designer, are systems in the ill-defined domain of graphic design (e.g., ACE: A Color Expert, an expert system for the selection of colors for synthetic scene imagery [Meier, 1986]; Descriptor, a generative system for graphic layout based on shape grammars [Glenn, 1986]). Some of these systems, like Designer, try to encode the general process of design and then apply it to a prototypical domain (e.g., PRIDE, for the design of paper handling

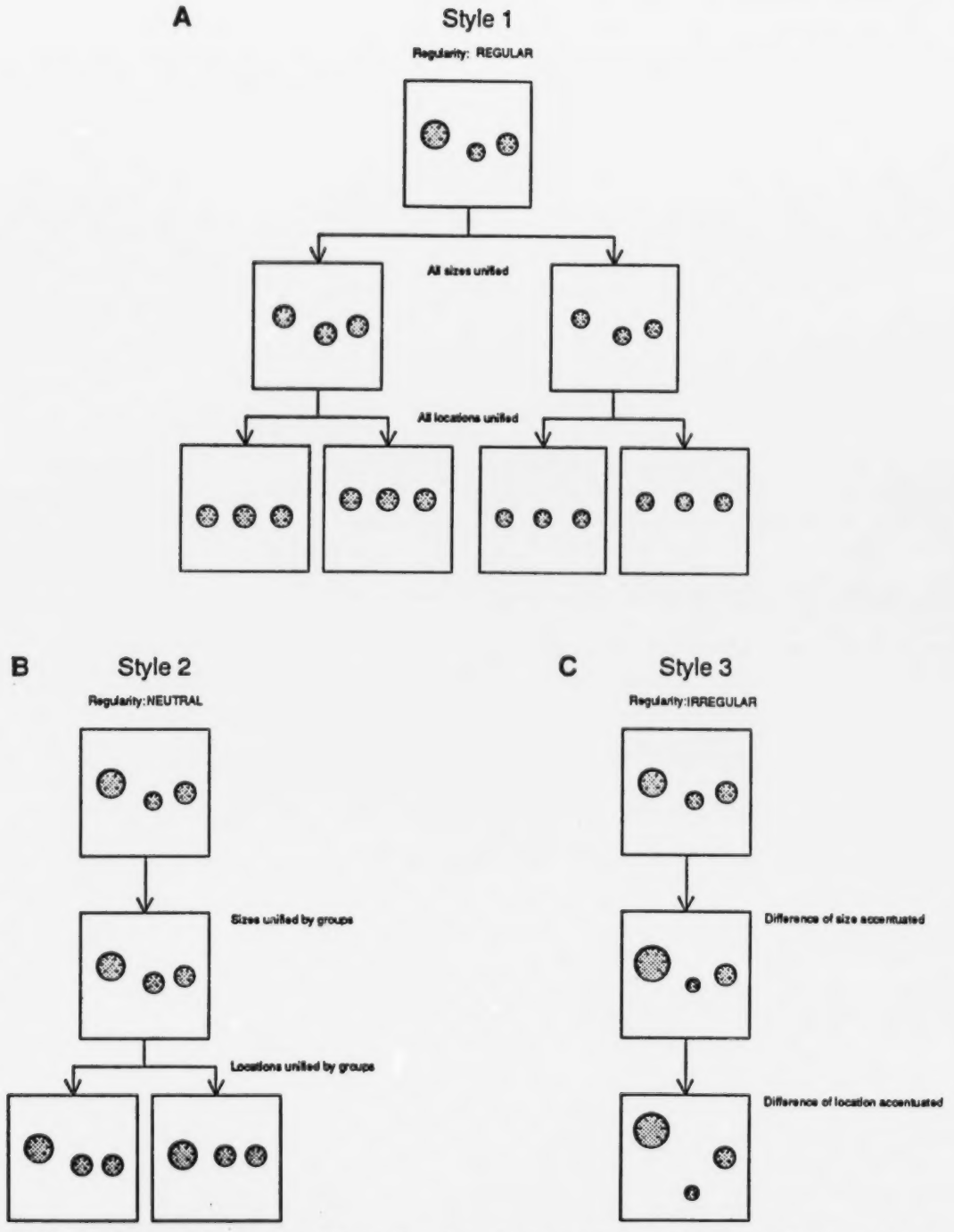

FIGURE 12. *Different Styles Generating Different Alternatives.* Given the same view as input, three different styles generate three completely different solutions. All three styles include the principles of Significant Difference of Size and of Location. They differ only in the articulation of the visual techniques defined, from simple and regular (A) to complex and irregular (C). The first level of design decisions is in response to element size while the second level is based on location considerations. These alternatives represent only a small portion of the solution space.

systems [Mittal et al., 1986]), while others use sophisticated tools (e.g., production systems and logic programming) on specific design problems. Designer differs slightly from these systems in that it is a *reactive* system. It responds to the users actions, by analyzing, critiquing, and interactively suggesting improvements to them. Currently, it does not try to create a new design given high-level design goals.

An initial implementation of Designer is underway. A functioning system has been used on existing Steamer diagrams and has provided useful feedback. It is very encouraging that even in views that were carefully crafted, the system was able to note inconsistencies and suggest improvements. In addition, a preliminary use of the ATMS has shown the feasibility of using it to maintain solutions in the design space.

The perception of a problem and the shape of its solution are both affected by the depth and range of the design vocabulary (Ching, 1979). Therefore, it is important that the domain knowledge base continues to grow. Only a few constraints currently exist and as more principles and standards are defined, more complete and robust solutions will be presented. As more solutions become available, better techniques to explore and understand their differences will be necessary.

ACKNOWLEDGMENTS

This project began as an Independent Exploratory Development Program (IED) with the Navy Personnel Research and Development Center in San Diego. Additional funding for this research is provided by the Personnel and Training Research Program of the Office of Naval Research (Contract N00014-85-C-0133, NR 667-541) and the Office of Naval Technology (522-801-018). I would like to thank all those involved with the UCSD/NPRDC joint program for making this research possible. In particular I would like to thank members of the group for their support *critiquing* early versions of this paper, including Jim Hollan, Mark Rosenstein, Dave Owen, Larry West, Barbara Morris, and Kathy Farrelly.

BIBLIOGRAPHY

Alexander, C. (1974). *Notes on the synthesis of form.* Cambridge, MA: Harvard University Press.

Beach, R., & Stone, M. (1983). Graphical style towards high quality illustrations. *Computer Graphics, 17,* 127-135.

Bertin, J. (1983). *Semiology of graphics.* Madison, WI: University of Wisconsin Press.

Borning, A. (1986). Defining constraints graphically. In M. Mantei & P. Orbeton (Eds.), *CHI'86 Proceedings* (pp. 137-143). New York: Association for Computing Machinery.

Brown, D. (1984). *Expert systems for design problem-solving using design refinement with plan selection and redesign.* Unpublished doctoral dissertation, Ohio State University.

Brown, D., & Chandrasekaran, B. (1986, July). Knowledge and control for a mechanical design expert system. *IEEE Computer,* pp. 92-100.

Brown, H., Tong, C., & Foyster, G. (1983, December). Palladio: An exploratory environment for circuit design. *IEEE Computer.*

Cheatham, F. R., Cheatham, J. H., & Haler, S. A. (1983). *Design concepts and applications.* Englewood Cliffs, NJ: Prentice-Hall.

Ching, F. D. K. (1979). *Architecture: Form, space, and order*. New York: Van Nostrand Reinhold.

De Kleer, J. (1986a). An assumption-based TMS. *Artificial Intelligence, 28*, 127-162.

De Kleer, J. (1986b). Extending the ATMS. *Artificial Intelligence, 28*, 163-196.

De Kleer, J. (1986c). Problem solving with the ATMS. *Artificial Intelligence, 28*, 197-224.

Dondis, D. A. (1973). *A primer of visual literacy*. Cambridge, MA: MIT Press.

Glenn, B. (1986). *Descriptor: A model for describing shapes that infers relations for positioning them.* Unpublished doctoral dissertation, University of California, Los Angeles.

Gullichsen, E., & Chang E. (1985). Generative design in architecture using an expert system. *Graphics Interface '85 Proceedings*. Canadian Information Processing Society.

Hollan J., Hutchins E., McCandless T., Rosenstein M., & Weitzman L. (in press). Graphical interfaces for simulations. In W. Rouse (Ed.), *Advances in man-machine systems research* (Vol. 3). Greenwich, CT: Jai Press.

Hollan, J., Hutchins, E., & Weitzman, L. (1984). Steamer: An interactive inspectable simulation-based training system. *AI Magazine, 5*(2), 15-28.

Hurlburt, A. (1977). *Layout: The design of the printed page*. New York: Watson-Guptill.

Marcus, A. (1986). Graphic design of user interfaces. *National Computer Graphics Association 1986 Proceedings: Vol. 1. Tutorials.*

McKinley, J. (1986). *Automatic design of graphical presentations*. Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Meier, B. (1986). ACE: A Color Expert. *National Computer Graphics Association 1986 Proceedings.*

Mittal, S., Dym, C., & Morjaria, M. (1986, July). PRIDE: An expert system for the design of paper handling systems. *IEEE Computer*, pp. 102-114.

Nelson, G. (1985). Juno: A constraint-based graphics system. *Computer Graphics, 19*, 235-243.

Pavlidis, T., & Van Wyk, C. J. (1986). An automatic beautifier for drawings and illustrations. *Computer Graphics, 19*, 225-234.

Reilly, S. S., & Roach, J. W. (1984, February). Improved visual design for graphics display. *IEEE Computer Graphics and Applications.*

Roach, J., Pittman, S.S., & Savarse, J. (1982, October). A visual design consultant. *Proceedings of the International Conference on Cybernetics and Society*, Seattle, WA.

Scholl, L. (1985). Heuristic rules for visualization. *Graphics Interface '85 Proceedings*. Canadian Information Processing Society.

Sherwood, R. (1981). *Principles and elements of architecture*. Los Angeles: University of Southern California Press.

Stiny, G., & Gips, J. (1978). *Algorithmic aesthetics: Computer models for criticism and design in the arts*. Los Angeles: University of California Press.

Taylor, I. A. (1960). Perception and visual communication. In J. Ball & F. Pyres, (Eds.), *Research principles and practices in visual communication*. Association for Education, Communication, and Technology.

Wong, W. (1972). *Principles of two-dimensional design*. New York: Van Nostrand Reinhold.

ICS Technical Report List

The following is a list of publications by people in the Institute for Cognitive Science. For reprints, write or call:

Institute for Cognitive Science, C-015
University of California, San Diego
La Jolla, CA 92093
(619) 452-6771

8301. David Zipser. *The Representation of Location.* May 1983.

8302. Jeffrey Elman and Jay McClelland. *Speech Perception as a Cognitive Process: The Interactive Activation Model.* April 1983. Also published in N. Lass (Ed.), *Speech and language: Volume 10*, New York: Academic Press, 1983.

8303. Ron Williams. *Unit Activation Rules for Cognitive Networks.* November 1983.

8304. David Zipser. *The Representation of Maps.* November 1983.

8305. The HMI Project. *User Centered System Design: Part I, Papers for the CHI '83 Conference on Human Factors in Computer Systems.* November 1983. Also published in A. Janda (Ed.), *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems.* New York: ACM, 1983.

8306. Paul Smolensky. *Harmony Theory: A Mathematical Framework for Stochastic Parallel Processing.* December 1983. Also published in *Proceedings of the National Conference on Artificial Intelligence, AAAI-83*, Washington DC, 1983.

8401. Stephen W. Draper and Donald A. Norman. *Software Engineering for User Interfaces.* January 1984. Also published in *Proceedings of the Seventh International Conference on Software Engineering*, Orlando, FL, 1984.

8402. The UCSD HMI Project. *User Centered System Design: Part II, Collected Papers.* March 1984. Also published individually as follows: Norman, D.A. (1984), Stages and levels in human-machine interaction, *International Journal of Man-Machine Studies, 21*, 365-375; Draper, S.W., The nature of expertise in UNIX; Owen, D., Users in the real world; O'Malley, C., Draper, S.W., & Riley, M., Constructive interaction: A method for studying user-computer-user interaction; Smolensky, P., Monty, M.L., & Conway, E., Formalizing task descriptions for command specification and documentation; Bannon, L.J., & O'Malley, C., Problems in evaluation of human-computer interfaces: A case study; Riley, M., & O'Malley, C., Planning nets: A framework for analyzing user-computer interactions; all published in B. Shackel (Ed.), *INTERACT '84, First Conference on Human-Computer Interaction*, Amsterdam: North-Holland,

1984; Norman, D.A., & Draper, S.W., Software engineering for user interfaces, *Proceedings of the Seventh International Conference on Software Engineering*, Orlando, FL, 1984.

8403. Steven L. Greenspan and Eric M. Segal. *Reference Comprehension: A Topic-Comment Analysis of Sentence-Picture Verification.* April 1984. Also published in *Cognitive Psychology, 16,* 556-606, 1984.

8404. Paul Smolensky and Mary S. Riley. *Harmony Theory: Problem Solving, Parallel Cognitive Models, and Thermal Physics.* April 1984. The first two papers are published in *Proceedings of the Sixth Annual Meeting of the Cognitive Science Society*, Boulder, CO, 1984.

8405. David Zipser. *A Computational Model of Hippocampus Place-Fields.* April 1984.

8406. Michael C. Mozer. *Inductive Information Retrieval Using Parallel Distributed Computation.* May 1984.

8407. David E. Rumelhart and David Zipser. *Feature Discovery by Competitive Learning.* July 1984. Also published in *Cognitive Science, 9,* 75-112, 1985.

8408. David Zipser. *A Theoretical Model of Hippocampal Learning During Classical Conditioning.* December 1984.

8501. Ronald J. Williams. *Feature Discovery Through Error-Correction Learning.* May 1985.

8502. Ronald J. Williams. *Inference of Spatial Relations by Self-Organizing Networks.* May 1985.

8503. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. *Direct Manipulation Interfaces.* May 1985. Also published in D. A. Norman & S. W. Draper (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction*, 1986, Hillsdale, NJ: Erlbaum.

8504. Mary S. Riley. *User Understanding.* May 1985. Also published in D. A. Norman & S. W. Draper (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction*, 1986, Hillsdale, NJ: Erlbaum.

8505. Liam J. Bannon. *Extending the Design Boundaries of Human-Computer Interaction.* May 1985.

8506. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Internal Representations by Error Propagation.* September 1985. Also published in D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1. Foundations*, 1986, Cambridge, MA: Bradford Books/MIT Press.

8507. David E. Rumelhart and James L. McClelland. *On Learning the Past Tense of English Verbs.* October 1985. Also published in J. L. McClelland, D. E. Rumelhart, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 2. Psychological and Biological Models*, 1986, Cambridge, MA: MIT Press/Bradford Books.

8601. David Navon and Jeff Miller. *The Role of Outcome Conflict in Dual-Task Interference*. January 1986.

8602. David E. Rumelhart and James L. McClelland. *PDP Models and General Issues in Cognitive Science*. April 1986. Also published in D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, 1986, Cambridge, MA: MIT Press/Bradford Books.

8603. James D. Hollan, Edwin L. Hutchins, Timothy P. McCandless, Mark Rosenstein, and Louis Weitzman. *Graphical Interfaces for Simulation*. May 1986. To be published in W. B. Rouse (Ed.), *Advances in Man-Machine Systems* (Vol. 3). Greenwich, CT: Jai Press.

8604. Michael I. Jordan. *Serial Order: A Parallel Distributed Processing Approach*. May 1986.

8605. Ronald J. Williams. *Reinforcement Learning in Connectionist Networks: A Mathematical Analysis*. June 1986.

8606. David Navon. *Visibility or Disability: Notes on Attention*. June 1986.

8607. William Appelbe, Donald Coleman, Allyn Fratkin, James Hutchison, and Walter J. Savitch. *Porting UNIX to a Network of Diskless Micros*. June 1986.

8608. David Zipser. *Programming Networks to Compute Spatial Functions*. June 1986.

8609. Louis Weitzman. *Designer: A Knowledge-Based Graphic Design Assistant*. July 1986.

Earlier Reports by People in the Cognitive Science Lab

The following is a list of publications by people in the Cognitive Science Lab and the Institute for Cognitive Science. For reprints, write or call:

Institute for Cognitive Science, C-015
University of California, San Diego
La Jolla, CA 92093
(619) 452-6771

ONR-8001. Donald R. Gentner, Jonathan Grudin, and Eileen Conway. *Finger Movements in Transcription Typing.* May 1980.

ONR-8002. James L. McClelland and David E. Rumelhart. *An Interactive Activation Model of the Effect of Context in Perception: Part I.* May 1980. Also published in *Psychological Review, 88,5*, pp. 375-401, 1981.

ONR-8003. David E. Rumelhart and James L. McClelland. *An Interactive Activation Model of the Effect of Context in Perception: Part II.* July 1980. Also published in *Psychological Review, 89, 1*, pp. 60-94, 1982.

ONR-8004. Donald A. Norman. *Errors in Human Performance.* August 1980.

ONR-8005. David E. Rumelhart and Donald A. Norman. *Analogical Processes in Learning.* September 1980. Also published in J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, NJ: Erlbaum, 1981.

ONR-8006. Donald A. Norman and Tim Shallice. *Attention to Action: Willed and Automatic Control of Behavior.* December 1980.

ONR-8101. David E. Rumelhart. *Understanding Understanding.* January 1981.

ONR-8102. David E. Rumelhart and Donald A. Norman. *Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance.* May 1981. Also published in *Cognitive Science, 6*, pp. 1-36, 1982.

ONR-8103. Donald R. Gentner. *Skilled Finger Movements in Typing.* July 1981.

ONR-8104. Michael I. Jordan. *The Timing of Endpoints in Movement.* November 1981.

ONR-8105. Gary Perlman. *Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System and MENUNIX: A Menu-Based Interface to UNIX (User Manual).* November 1981. Also published in *Proceedings of the 1982 USENIX Conference*, San Diego, CA, 1982.

ONR-8106. Donald A. Norman and Diane Fisher. *Why Alphabetic Keyboards Are Not Easy to Use: Keyboard Layout Doesn't Much Matter.* November 1981. Also published in *Human Factors, 24*, pp. 509-515, 1982.

ONR-8107. Donald R. Gentner. *Evidence Against a Central Control Model of Timing in Typing.* December 1981. Also published in *Journal of Experimental Psychology: Human Perception and Performance, 8*, pp. 793-810, 1982.

ONR-8201. Jonathan T. Grudin and Serge Larochelle. *Digraph Frequency Effects in Skilled Typing*. February 1982.

ONR-8202. Jonathan T. Grudin. *Central Control of Timing in Skilled Typing*. February 1982.

ONR-8203. Amy Geoffroy and Donald A. Norman. *Ease of Tapping the Fingers in a Sequence Depends on the Mental Encoding*. March 1982.

ONR-8204. LNR Research Group. *Studies of Typing from the LNR Research Group: The role of context, differences in skill level, errors, hand movements, and a computer simulation.* May 1982. Also published in W. E. Cooper (Ed.), *Cognitive aspects of skilled typewriting*. New York: Springer-Verlag, 1983.

ONR-8205. Donald A. Norman. *Five Papers on Human-Machine Interaction.* May 1982. Also published individually as follows: Some observations on mental models, in D. Gentner and A. Stevens (Eds.), *Mental models*, Hillsdale, NJ: Erlbaum, 1983; A psychologist views human processing: Human errors and other phenomena suggest processing mechanisms, in *Proceedings of the International Joint Conference on Artificial Intelligence*, Vancouver, 1981; Steps toward a cognitive engineering: Design rules based on analyses of human error, in *Proceedings of the Conference on Human Factors in Computer Systems*, Gaithersburg, MD, 1982; The trouble with UNIX, in *Datamation, 27,12*, November 1981, pp. 139-150; The trouble with networks, in *Datamation*, January 1982, pp. 188-192.

ONR-8206. Naomi Miyake. *Constructive Interaction.* June 1982.

ONR-8207. Donald R. Gentner. *The Development of Typewriting Skill.* September 1982. Also published as Acquisition of typewriting skill, in *Acta Psychologica, 54*, pp. 233-248, 1983.

ONR-8208. Gary Perlman. *Natural Artificial Languages: Low-Level Processes.* December 1982. Also published in *The International Journal of Man-Machine Studies, 20*, pp. 373-419, 1984.

ONR-8301. Michael C. Mozer. *Letter Migration in Word Perception.* April 1983. Also published in *Journal of Experimental Psychology: Human Perception and Performance, 9, 4*, pp. 531-546, 1983.

ONR-8302. David E. Rumelhart and Donald A. Norman. *Representation in Memory.* June 1983. To appear in R. C. Atkinson, G. Lindzey, & R. D. Luce (Eds.), *Handbook of experimental psychology*. New York: Wiley (in press).